View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Saint Petersburg State University

Санкт-Петербургский государственный университет

Математическое обеспечение и администрирование информационных систем
Информационно-аналитические системы

Рафикова Элона Рустамовна

Алгоритмы индексирования для памяти PCM

Бакалаврская работа

Научный руководитель:
д. ф.-м. н., профессор Новиков Б. А.

Рецензент:
инженер Смирнов К. К.

Санкт-Петербург
2017

SAINT-PETERSBURG STATE UNIVERSITY

Information Systems Administration and Mathematical Support
Analytical Information Systems

Rafikova Elona

Indexing algorithms for PCM memory

Graduation Thesis

Scientific supervisor:
professor Boris Novikov

Reviewer:
engineer Kirill Smirnov

Saint-Petersburg
2017

Оглавление

Введение

Самые разные вычислительные системы немыслимы без использования устройств хранения. Но до сих пор нет универсального типа памяти, который удовлетворял бы всем критериям. Большая разница в производительности между запоминающими устройствами, а также все новые и новые требования от электронных систем, например портативных электронных устройств, открывают новые возможности для устройств хранения.

Память с изменением фазового состояние (Phase Change Memory, сокращенно PCM) - одно из многообещающих устройств хранения. Это новый физический принцип хранения информации, который работает посредством изменения фазового состояния халькогенидного стекла, из которого состоят PCM-ячейки. Различие в этих состояниях, а именно разное электрическое сопротивление, используется для кодирования информации. Считывание состояния происходит благодаря использованию сверхмалых токов, которые не оказывают воздействие на вещество в ячейке. Для перевода ячейки из одного состояния в другое необходимо использовать более сильные токи, которые изменяют фазовое состояние посредством нагрева ячейки.

В первую очередь целевые продукты для PCM - это мобильные телефоны, но в будущем ожидается, что PCM будет компонентом в иерархии памяти для ноутбуков, персональных компьютеров и серверов. От использования PCM выиграют и центры обработки данных, которые смогут быстрее обрабатывать огромные массивы информации, связанные с "облачными" вычислениями, финансовыми транзакциями и машинным обучением. Память с изменением фазового состояния соединяет в себе такие свойства DRAM, как побитовое изменение, быстрое чтение и запись, а также энергонезависимость и простую структуру, как у Flash памяти. Но в то же время память PCM обладает ограничением на количество перезаписи, хотя это ограничение на порядок лучше чем у Flash. Комбинация этих свойств делает PCM уникальным видом памяти следующего поколения с расширенным набором возможностей

[3].

Тогда совершенно естественно изучать перспективный вид памяти и одним из главных вопросов становится: Как имеющиеся методы работы с данными должны быть изменены, чтобы наилучшим образом воспользоваться преимуществами PCM и, по возможности, сгладить недостатки?

В этой работе предложен модифицированный метод расширяемого хеширования [6], который является дружественным к PCM, то есть является более эффективным для памяти с изменением фазового состояния, чем обычный. Также был проведен ряд экспериментов, доказывающих этот факт.

1. Задача построения индексных структур для PCM

Память с изменением фазового состояния - относительно новый вид памяти, и еще только рассматривается в качестве главного кандидата на роль универсальной памяти в компьютерах и системах хранения данных. Поэтому очень важно для эффективного применения изучить ее особенности.

Кроме того, не менее значимым является изучение алгоритмов, работающих с памятью, например, такая категория, как алгоритмы индексирования, которые абсолютно необходимы для быстрого поиска информации в базах данных, которые в свою очередь является неотъемлемой частью почти любой информационной системы.

Целью данной работы является исследование возможности применения PCM для создания новых методов индексирования. Для достижения этой цели были поставлены следующие задачи:

- Изучить характеристики и свойства PCM
- Сравнить PCM с другими типами памяти
- Рассмотреть существующие методы индексирования и их применимость для PCM
- На основе изученных исследований предложить модификацию алгоритма хеширования
- Провести ряд экспериментов и проанализировать полученные результаты

2. Сравнение типов памяти

Для того, чтобы работать с памятью нового типа, необходимо узнать ее характеристики, а также сравнить с уже существующими и широко используемыми типами памяти.

Рассмотрим таблицу, позволяющую сравнить различные виды памяти по параметрам:

	DRAM	PCM	NAND Flash
Retention	Refresh	10 years	10 years
Density	1×	2-4×	4×
Endurance	$\sim 10^{15}$	10^6-10^8	10^4-10^5
Page size	64 B	64 B	4 KB
Read latency	20-50 ns	~50 ns	~25 μs
Write latency	20-50 ns	~1 μs	~500 μs
Erase latency	N/A	N/A	~2 ms /block
Write BW /die	~1 GB/s	50-100 MB/s	5-40 MB/s
Read energy	0.8 J/GB	1 J/GB	1.5 J/GB
Write energy	1.2 J/GB	6 J/GB	17.5 J/GB
Idle power	~100 mW/GB	~1 mW/GB	1-10 mW/GB

Рис. 1: Сравнение технологий памяти: PCM, DRAM и NAND flash [3]

Из таблицы на Рис.1 видно, что характеристики PCM находятся между DRAM и NAND. Можно сказать, что PCM считается многообещающей альтернативой DRAM, в качестве основной памяти, или перспективной заменой NAND Flash, в качестве запоминающего устройства.

По сравнению с NAND, PCM обладает рядом преимуществ по производительности, энергосбережению и выносливости.

Но для алгоритмов индексирования интереснее рассматривать PCM в качестве основной памяти. А по сравнению с DRAM, PCM может обеспечить следующие преимущества: 1) энергонезависимость, которая позволяет хранить данные даже при выключенном питании; 2) большую плотность, в 2-4 раза больше, что подразумевает большую емкость памяти с той же площадью кристалла; 3) сравнимое время ожидания чтения и потребление энергии с DRAM, и 4) почти нулевое энергопотребление.

Кроме того, считается что, память DRAM восприимчива к случайным сбоям, которые вызываются альфа-частицами или космическим

излучением. В PCM этот эффект не наблюдается.

Несмотря на эти привлекательные особенности, PCM имеет несколько недостатков: 1) проблема изнашиваемости, которая означает, что каждая ячейка будет изнашиваться после ограниченного числа операций записи; 2) время записи примерно в 20 раз больше, чем операция чтения в PCM или чтение/запись в DRAM; 3) более высокое потребление энергии для записи в память.

К настоящему времени проведена большая исследовательская работа по предотвращению отрицательного влияния недостатков PCM, чтобы сделать замену основной памяти на PCM осуществимой [1, 9, 5, 4, 2]. Некоторые улучшения увеличивают срок службы PCM с помощью стратегий выравнивания износа и сокращения избыточных операций записей. Другие же решают длительную задержку записи с использованием DRAM-буфера на уровне архитектуры.

Далее будут рассмотрены улучшения, связанные с уменьшением количества операций записи на уровне логики алгоритмов, для сокращения потребляемой энергии и увеличения времени эксплуатации систем хранения на основе PCM.

3. Запись в PCM

Одной из основных проблем эффективного использования PCM являются упомянутые выше ограничения, связанные с записью данных в память.

По сравнению с чтением ячейки PCM, операция записи потребляет более высокий ток, использует более высокое напряжение и занимает больше времени из-за физических свойств памяти. Получается, что операция записи потребляет в 6-10 раз больше энергии, чем чтение.

В PCM устройстве задержка записи в ячейке примерно в три раза больше, чем при операции чтения. Более того, многие прототипы такого вида памяти поддерживают итеративную запись, то есть ограниченное число битов на одну итерацию, для того чтобы сократить мгновенный уровень тока. В будущем это явление скорее всего сохранится, особенно для систем с небольшими физическими ресурсами. Тогда из-за ограниченной пропускной способности получается еще большая задержка записи.

Существующие прототипы предоставляют возможность делать ограниченное количество перезаписи в ячейку, а именно от 10^6 до 10^8 операций записи на ячейку. Получается, что PCM, даже при хороших алгоритмах выравнивания износа, как основная память, может сохраняться только в течении нескольких лет при реальных нагрузках.

Исходя из вышесказанного, основной задачей при разработке алгоритмов, дружественных к PCM, является преодоление асимметрии между PCM-чтением и PCM-записью. Поэтому одной из целей разработки эффективных алгоритмов для PCM является минимизация количества операций записи.

4. Методы индексирования на основе хеширования

Как известно, хеш-таблицы обеспечивают функциональность ассоциативного массива, сохраняя пары "ключ-значение" в определенных местах (корзинах), которые определяются путем применения одной или нескольких хеш-функций к ключу.

Алгоритмы хеширования широко используются в базах данных для индексации, а также в большинстве других случаев, когда требуется быстрый поиск. В связи с этим, уже были проведены исследования, в которых анализировалась применимость для PCM основных видов статического хеширования: с цепочками, с открытой адресацией, хеширование массивом, также их модификации: "кукушка" (Cuckoo) и "классики" (Hopscotch) [10].

Хеширование "кукушка", использует массив для пар "ключ-значение". Каждая пара может быть помещена в одну из двух корзин. Хеширование "кукушка" использует две отдельные хеш-функции для вычисления этих двух корзин-адресов. Если во время вставки обе корзины заняты, пара в одной из двух корзин случайно выбирается и помещается в другую корзину, чтобы освободить место для входящей пары. Кроме того, смещенная пара может вытеснить другую пару, если обе ее корзины заполнены, создавая последовательность смещений. Этот процесс продолжается до тех пор, пока не потребуется дальнейшее перемещение (то есть будет найдена пустая корзина). Когда пару уже никуда нельзя сместить, хеш-таблица объявляется полной и требуется изменение размера [8].

Другой алгоритм хеширования "классики" - это модификация метода "кукушка". При таком алгоритме хеширования все возможные места для заданного ключа являются смежными. Количество этих местоположений называется расстоянием "прыжка". Для ускорения поиска каждая корзина хранит битовую шкалу, которая указывает позиции в окрестности, которые необходимо исследовать, чтобы найти элемент. Во время вставки элемент хешируется в определенное местоположение

на основе ключа. Затем он использует линейное пробирование, чтобы найти пустую корзину. Если пустая корзина выходит за пределы диапазона "прыжка", предпринимается попытка переместить другие элементы из этого диапазона на места которые они могут занять, освободив тем самым место для вставляемого элемента. Если такое перемещение невозможно, хеш-таблица становится полной и требует реорганизации с изменением размера [7].

На Рис. 2 показано сравнение существующих конструкций хеш-таблиц.

Collision Resolution Strategy	CPU Cache Utilization	Cascading Writes	Frequent Memory Allocation
Chaining	Low	No	Yes
Arrayhash	High	No	Yes
Linear Probing	Medium	No	No
Cuckoo	Low	Yes	No
Hopscotch	High	Yes	No
PFHT (our design)	High	No	No

Рис. 2: Сравнение хеш-таблиц [10]

Хеширование с цепочками, которое использует связанные списки для хранения коллизирующих элементов, использует динамическое выделение памяти и доступ через указатели, что приводит к большому количеству промахов кэша центрального процессора (ЦП), снижая производительность.

Хеширование массивом использует непрерывную память, чтобы избежать проблемы с указателями. Однако этот метод все еще использует понижающие производительность частые выделения памяти.

Линейное пробирование улучшает использование кэша ЦП, избегая динамического выделения памяти и доступа к элементам в последовательных линиях кэша. Однако, по мере увеличения коэффициента загрузки таблицы, вставка одной пары требует прохождения большого количества строк кэша, чтобы найти пустое место.

Хеширования "кукушка" и "классики" не требуют динамического распределения памяти и поддерживают высокий коэффициент загрузки с ограниченным временем поиска. Однако эти хеш-таблицы страда-

ют от проблемы каскадных записей. Под каскадной записью имеется в виду, что для вставки пары нам может потребоваться сместить существующие пары, что приведет к множественным операциям записи для одной вставки.

На Рис.3 показан пример каскадной записи.

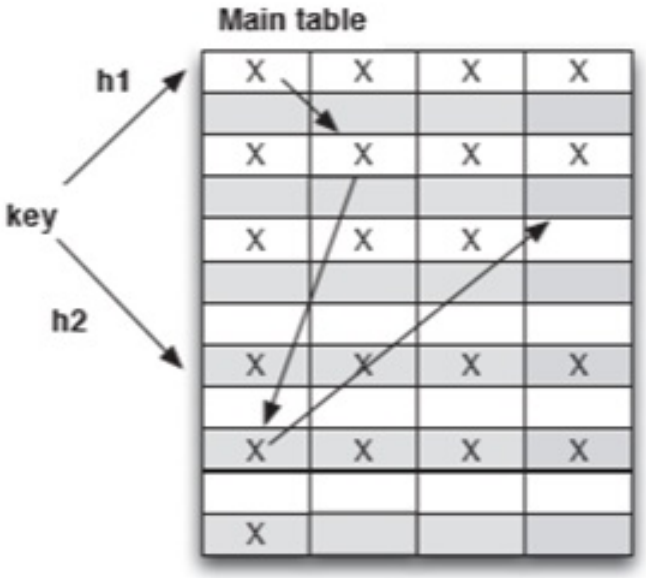


Рис. 3: Каскадная запись при хешировании ”кукушка” [3]

В системах на основе DRAM существует симметрия между производительностью операций чтения и записи. Как результат, производительность таблиц над DRAM в основном зависит от использования кэша ЦП. Однако для систем на базе PCM асимметрия производительности чтения и записи подразумевает, что количество записей дополнительно влияет на производительность хеш-таблицы. Каскадные записи, вызванные смещениями пары во время вставки, замедляют обновление и быстро изнашивают PCM.

Что касается каскадных записей, хеширование с цепочками, хеширование массивом и линейное пробирование работают лучше (особенно при большом коэффициенте загрузки таблицы), в то время как хеширование ”кукушка” и ”классики” хуже. Что касается использования кэша ЦП, то хеширование с цепочками и кукушка происходят хуже, чем ли-

нейное пробирование и "классики". Исходя из всего вышесказанного, был предложен вариант (PCM Friendly Hash Table, сокращенно PFHT) [3], как создать эффективно работающую с памятью PCM хеш-таблицу, которая не страдает от неограниченного времени поиска, например, как линейное пробирование, предоставляя при этом преимущества многоуровневых конструкций, таких как "кукушка".

PFHT - это модификация алгоритма хеширования "кукушка", который сохраняет его преимущества, то есть эффективное использование памяти при высоком коэффициенте загрузки таблицы, а также гарантированное время поиска значения. Тем не менее, PFHT избегает проблем каскадных записей, которые влияют на производительность.

Этот метод использует два компонента - основную таблицу (Main table) и "тайник" (Stash), как показано на Рис. 4. Основная таблица обеспечивает хорошую производительность кэша процессора и позволяет избежать каскадных записей, в то время как "тайник" помогает PFHT достичь высокого коэффициента загрузки.

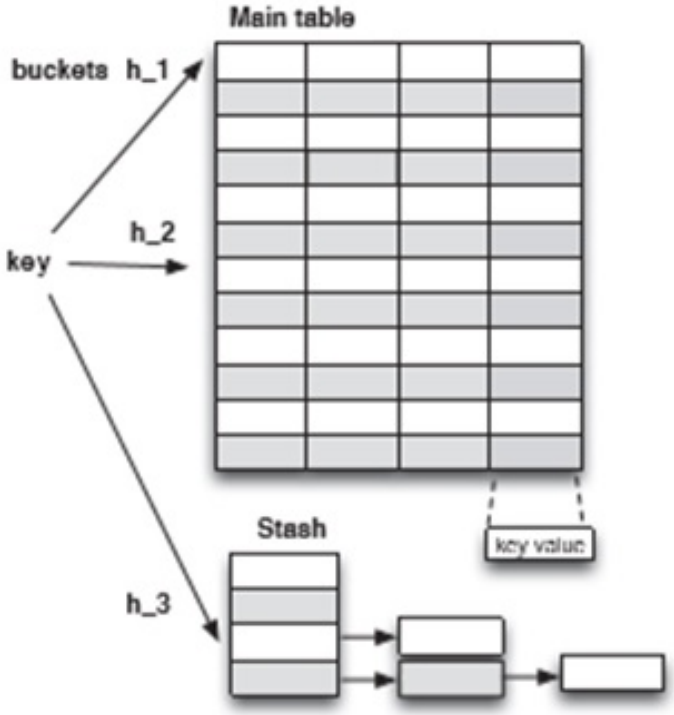


Рис. 4: Схема PFHT [3]

Основная таблица реализуется как двумерный массив корзин. Корзины хранятся в памяти непрерывно. Каждая корзина содержит фиксированное количество мест для хранения пар "ключ-значение". PFHT использует два варианта для хранения любой входящей пары. Кроме того, каждая корзина заполняется одной или несколькими последовательными линиями кэша. Следовательно, количество мест в корзине зависит от размера строки кэша, который достаточно велик в современных процессорах.

"Тайник" - небольшое вспомогательное хранилище, которое PFHT использует для хранения любых пар, которые он не может вставить в основную таблицу, повышая при этом коэффициент загрузки.

PFHT обладает следующими свойствами, которые делают его привлекательным выбором для PCM:

- Сбалансированная вставка, а именно: пара может храниться в одной из двух возможных корзин в главной таблице. PFHT выбирает наименее загруженную корзину для хранения пары и сохраняет таблицу сбалансированной. В отличие от этого, хеширование "кукушка" выбирает корзину случайным образом, чтобы вставить пару "ключ-значение". Эта схема не оказывает неблагоприятного воздействия на DRAM даже при неограниченной длине цепи перемещений. Однако дополнительные записи вредят производительности PCM. Путем сохранения балансировки PFHT гарантирует, что дополнительные записи не требуются для вставки новых пар.

- Отсутствие каскадных записей: PFHT избегает каскадного эффекта записи, допуская только одно перемещение. Во время вставки, если обе корзины заняты, PFHT проверяет, можно ли переместить любую пару на занятом месте в альтернативное месторасположение. Если это невозможно, то вставляемая пара перемещается в "тайник".

- Высокая загрузка кэша ЦП: PFHT хранит ключи в одном месте в смежных корзинах, что улучшает производительность кэша

ЦП. Для достижения высокого коэффициента нагрузки PFHT использует большой размер корзин. Однако использование больших размеров корзин увеличивает время ожидания поиска. Поэтому, чтобы добиться хорошего баланса между коэффициентом загрузки и времени поиска, PFHT устанавливает размер корзины в две строки кэша.

Из вышесказанного следует, что PFHT избегает проблемы каскадных записей в хеш-таблицах и обеспечивает хорошую производительность при больших рабочих нагрузках, что было подтверждено экспериментами.

5. PCMFEH

Авторы существующих решений на тему применимости разного вида хеш-таблиц для памяти типа PCM не рассматривали алгоритм расширяемого хеширования.

Расширяемое хеширование часто используется в базах данных, так как базы данных могут быть крайне большими и перехеширование всей базы данных займет продолжительное время, при этом лишая пользователей доступа к ней. А при использовании этого алгоритма перехешировать придется только малые группы, что не сильно замедлит работу базы данных.

Далее расмматривается и описывается модификация алгоритма расширяемого хеширования, предложенная в этой работе (PCMFEH - PCM friendly extensible hashing). Эта модификация учитывает особенности памяти с изменением фазового состояния.

Метод расширяемого хеширования заключается в том, что хеш-таблица представлена как каталог, каждая ячейка которого указывает на страницы в нем. А также имеется определенный алгоритм вставки пары "ключ-значение". Сама хеш-таблица будет иметь глобальную глубину, а каждая из страниц имеет локальную глубину. Глобальная глубина показывает сколько последних бит будут использоваться для хеш-функции. А из разницы локальной глубины и глобальной глубины можно понять сколько ячеек каталога ссылаются на страницу. Это можно показать формулой $K = 2^{(G-L)}$ где G - глобальная глубина, L - локальная глубина, а K - количество ссылающихся ячеек.

В основу измененного алгоритма легла идея о допустимости переполнения. А именно: структура хранения данных модифицируется так, что страница может иметь на одну или несколько пар "ключ-значение" больше, чем позволяет максимальный размер страницы.

Для реализации такой идеи введем два новых понятия:

1. Коэффициент переполнения - это максимальное количество пар, которые можно вставить в уже полностью заполненную страницу. (обозначим OVF)

2. Глубина переполнения страницы - это разность между максимальным размером страницы и фактическим.(Показывает, на сколько пар переполнена страница)

Тогда модифицированный алгоритм добавления пары такой:

1. Переводим ключ в двоичный вид, смотрим на последние G битов и решаем, в какую страницу поместить значение.

2. Если страница имеет свободное место, то помещаем туда значение, если же страница, куда следует положить значение переполнена, то смотрим на глубину переполнения:

   (a) Если она меньше, чем коэффициент переполнения, значит в данную страницу можно еще поместить значение, помещаем его и увеличиваем глубину переполнения на 1.

   (b) Если же глубина переполнения равна коэффициенту переполнения, то значит следует смотреть на локальную глубину:

      i. Если она меньше, чем глобальная глубина, то значит на страницу есть несколько указателей и нам достаточно перехешировать ее, разделив при этом на две и занести значения в новые две страницы, увеличив их локальную глубину на 1, и изменить значение глубины переполнения на разность между суммой максимального размера страницы с коэффициентом переполнения и фактическим размером страницы.

      ii. Если же локальная глубина была равна глобальной, то мы увеличиваем глобальную глубину на 1, удваивая при этом количество ячеек в каталоге, количество указателей на страницы, а также увеличиваем количество последних бит, по которым мы распределяем значения. Далее локальная глубина переполненной страницы становится меньше, чем глобальная глубина и мы выполняем предыдущий шаг, то есть перехешируем нужную страницу, разделим ее на две страницы и так далее.

6. Сравнение расширяемого хеширования и PCMFEH

Для исследования расширяемого хеширования был реализован алгоритм на языке C++ в среде разработки Visual Studio, с возможностью добавления пары "ключ-значение", удаления и поиска по ключу.

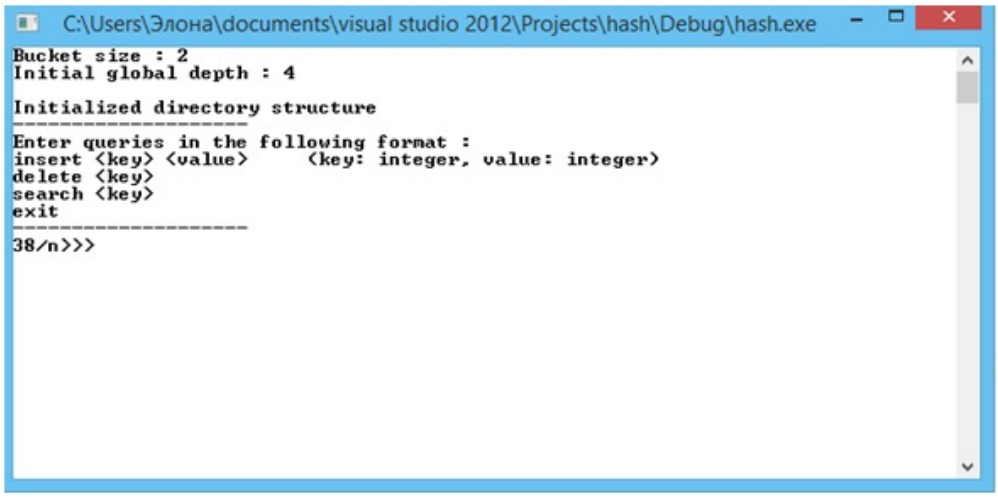


Рис. 5: Интерфейс реализации алгоритма расширяемого хеширования

Так как основная проблема памяти PCM - это неэффективность операции записи (описано в разделе Запись в PCM), стоит проверить, какое количество операций записи будет при различных действиях в хеш-таблице.

В этом методе хеширования мы можем менять начальную глобальную глубину (G), а так же размер страницы (BS).

В первом эксперименте проверим, сколько операций записи будет выполнено при построении структуры для хранения в стандартном расширяемом хешировании. (Здесь и далее, каждый эксперимент повторялся 20 раз, в качестве результата бралось среднее значение)

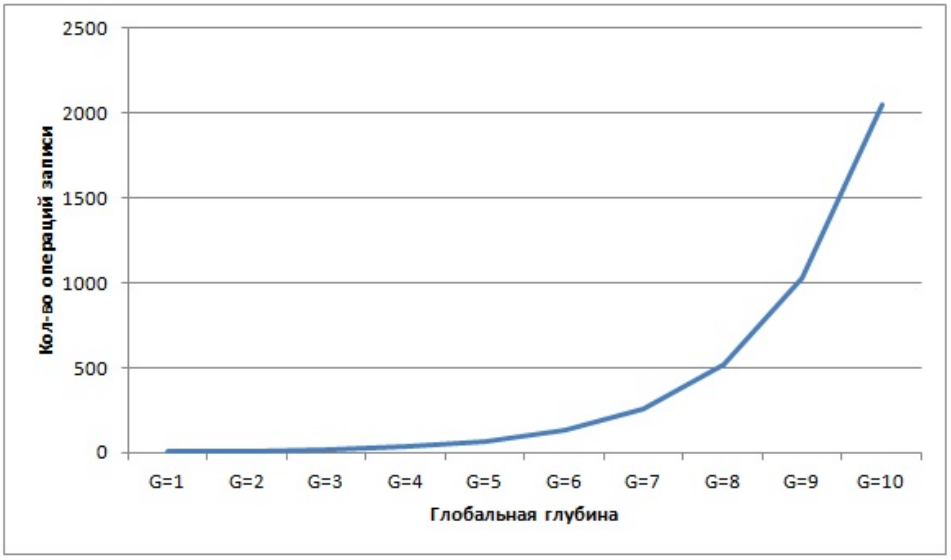


Рис. 6: Зависимость количества операций записи от глобальной глубины (G) при построении структуры хранения

Получается ожидаемый результат: при возрастании глубины возрастает и количество операций, так как глубина непосредственно влияет на начальную структуру для хранения данных. Если глубина равна G, то количество страниц в структуре будет равно 2^G. Таким образом, при построении структуры необходимо учитывать рост количества операций от глобальной глубины.

В следующем эксперименте для расширяемого хеширования будем изменять размер страниц, ведь он влияет на число коллизий, а следовательно на число расщеплений страниц и перераспределение значений. То есть непосредственным образом на число операций записи. Проверим это. Будем вставлять по 1000 пар, где и ключи и значения будут числами от 0 до 100 000.

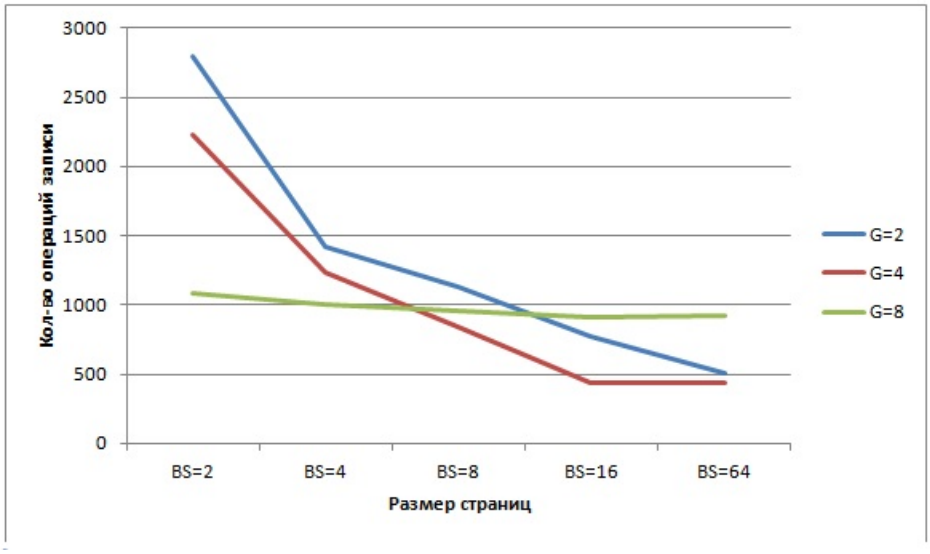


Рис. 7: Зависимость количества операций записи от размера страниц (BS) при разной глобальной глубине (G)

В результате получаем, что для глубины G=2 график наиболее сильно убывает, это связано с тем, что изначально структура имеет 4 страницы, и очевидно, что при их малом размере требуется очень много перестроений, а если страницы большие, то перестроений значительно меньше. При G=8, мы имеем 512 страниц, так что вне зависимости от их размера, вероятность коллизий очень мала, и поэтому график почти постоянный. При G=4 мы имеем наилучшие результаты, график так же как и при G=2 заметно убывает, а при размерах страниц 8 и 16 результаты наилучшие.

Исходя из этого, можно сделать вывод, что при использовании такого хеширования нужно находить баланс между начальной глубиной и размером страницы.

Для уменьшения количества операций нужно или увеличивать глубину, а следовательно и количество страниц, для уменьшения перестроений, но тогда стоит учитывать расходы на построения, которые примерно равны $2^{(G+1)}$(из первого эксперимента), или увеличивать размер страницы, что не лучший вариант для хеширования, потому что при поиске или удалении значения, потребуется больше времени для нахождения элемента в большой странице.

На основе информации, полученной из экспериментов, будем сравнивать расширяемое хеширование и его модификацию PCMFEH.

А именно, сначала сравним количество операций при построении структуры. В модифицированном методе количество операций должно вырасти за счет добавлений коэффициента переполнения и глубины переполнения.

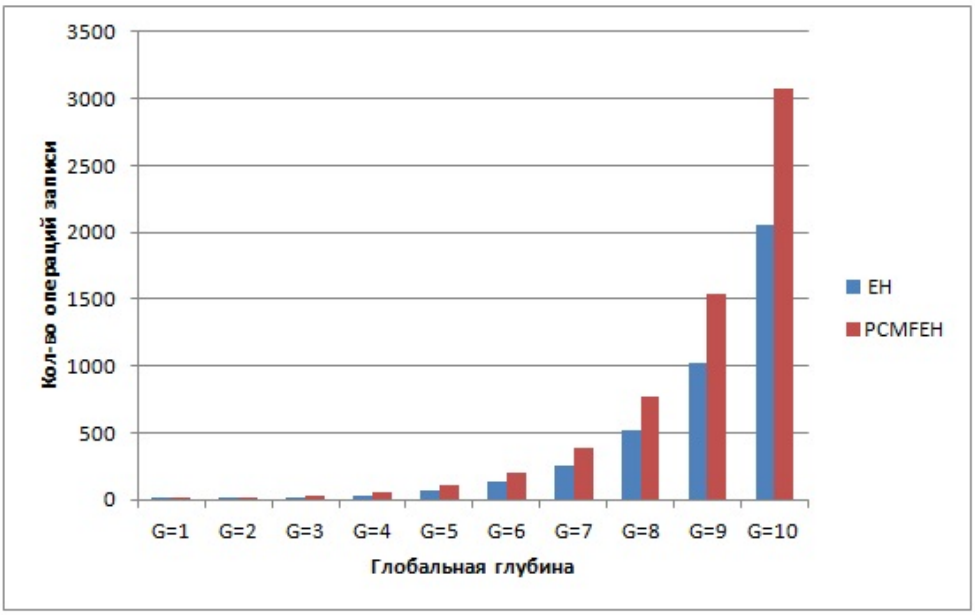


Рис. 8: Сравнение алгоритма расширяемого хешированиям (EH) и модифицированного метода (PCMFEH). Зависимость количества операций записи при построении структуры от глобальной глубины (G)

Как и предполагалось, у PCMFEH количество операций записи больше, чем у обычного расширяемого хеширования, особенно эта разница заметна при глобальной глубине начиная с 8. Но, так как использования большого значения глобальной глубины целесообразно при большом количестве вставляемых пар, даже такая разница не повлияет на производительность модифицированного метода, так как это разница будет несущественна по сравнению с общим количеством операций записи. Этот факт будет далее подтвержден экспериментами.

Теперь сравним модифицированный алгоритм при разных коэффициентах переполненности. Будем вставлять по 1000 пар, где ключи и значения будут числами от 0 до 100 000.

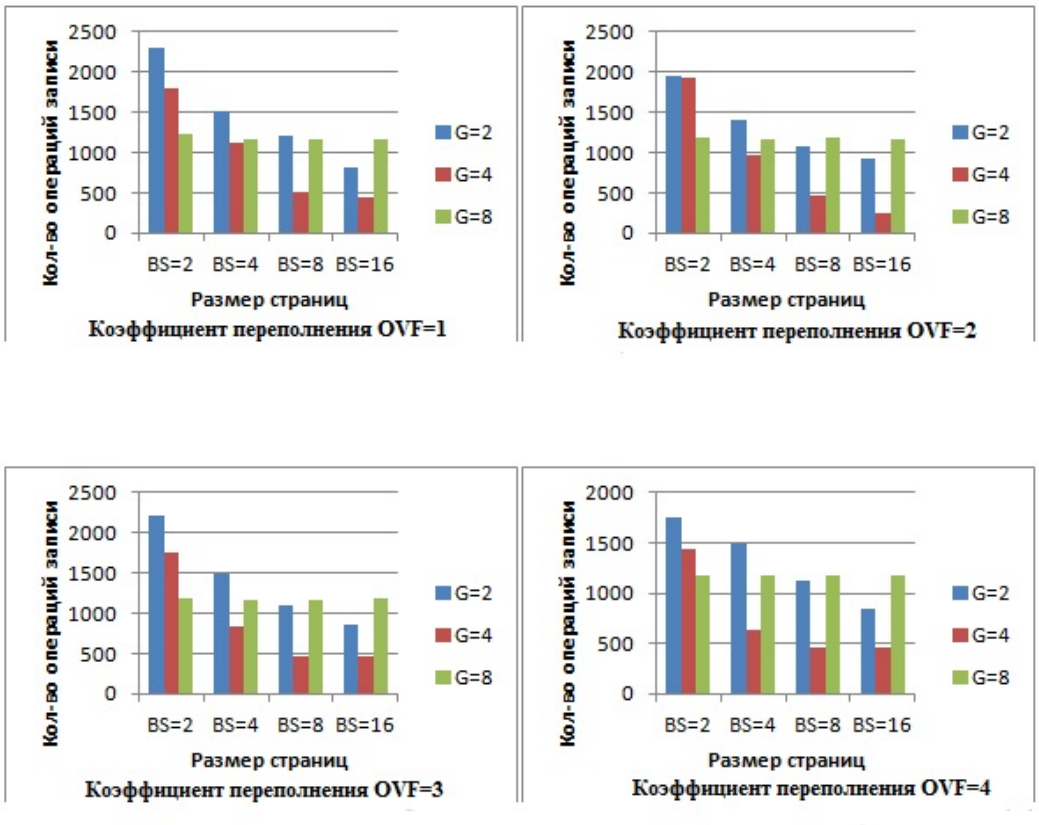


Рис. 9: Сравнение PCMFEH при разных коэффициентах переполнения (OVF)

Мы получаем результат, что количество операций записи при глобальной глубине G=8 почти не изменяется, это связано с тем, что, при любом размере страницы из большого количества страниц и возможности переполнения, перестроений происходило крайне мало.При G=2 результаты все так же велики, потому что из-за малого количества страниц перехеширования неизбежны даже с допущением переполнения. А глобальная глубина G=4 предоставляет достаточное количество страниц и, добавляя к этому возможность переполнения, получаются хорошие результаты. Также если сравнивать методы с разным коэффициентом переполнения, мы видим наилучшие показатели при OVF=3 и OVF=4. Но опять же, при выборе коэффициента переполнения, стоит помнить, что, увеличивая его, мы тем самым увеличиваем фактический размер страницы, а следовательно и время поиска ключа в странице.

Теперь приступим к непосредственному сравнению обычного метода

расширяемого хеширования и модифицированного. Будем также вставлять по 1000 пар, в которых ключи и значения будут числами от 0 до 100 000, значение глобальной глубины рассмотрим при G=2 и G=4, размер страницы будем изменять от BS=2 до BS=16.

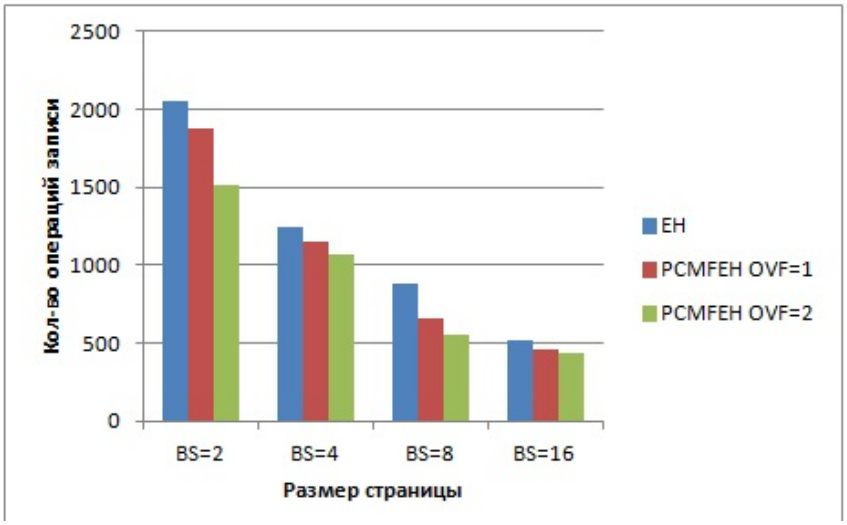


Рис. 10: Сравнение расширяемого хеширования (EH) с PCMFEH с коэффициентами переполнения 1 и 2 (Глобальная глубина G=2)

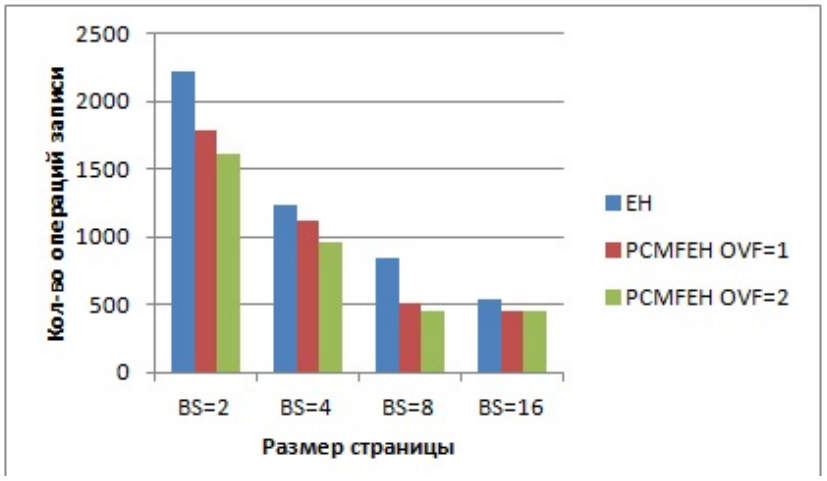


Рис. 11: Сравнение расширяемого хеширования (EH) с PCMFEH с коэффициентами переполнения 1 и 2 (При глобальной глубине G=4)

Показанные эксперименты подтверждают теоретическое предположение о сокращении количества операций записи в алгоритме PCMFEH. Уменьшение происходит при любой глубине и размере страницы.

Далее следует проверить, не будет ли отрицательно влиять эта модификация на время поиска значений.

Для этого будем искать 20 существующих пар в заполненных структурах и замерять время поиска.

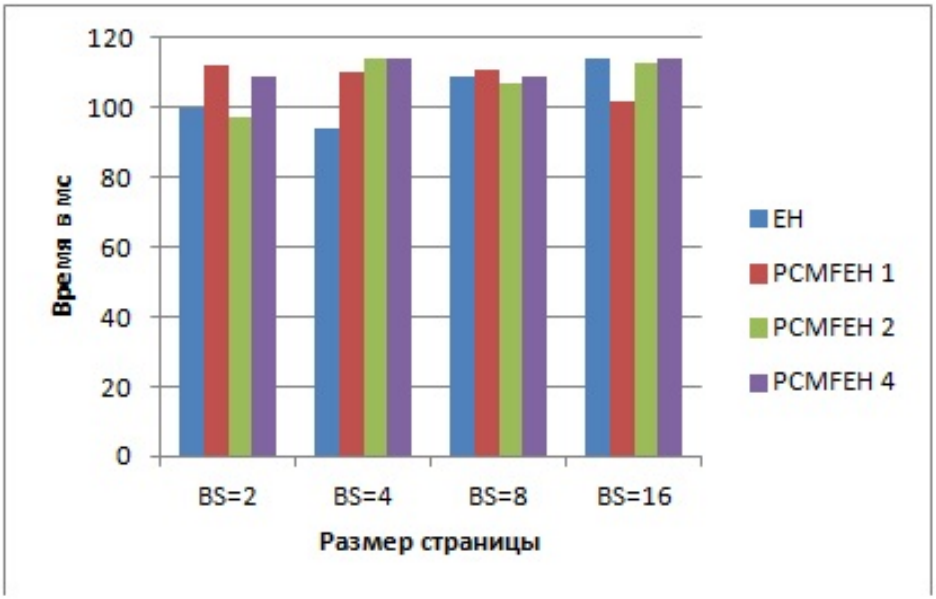
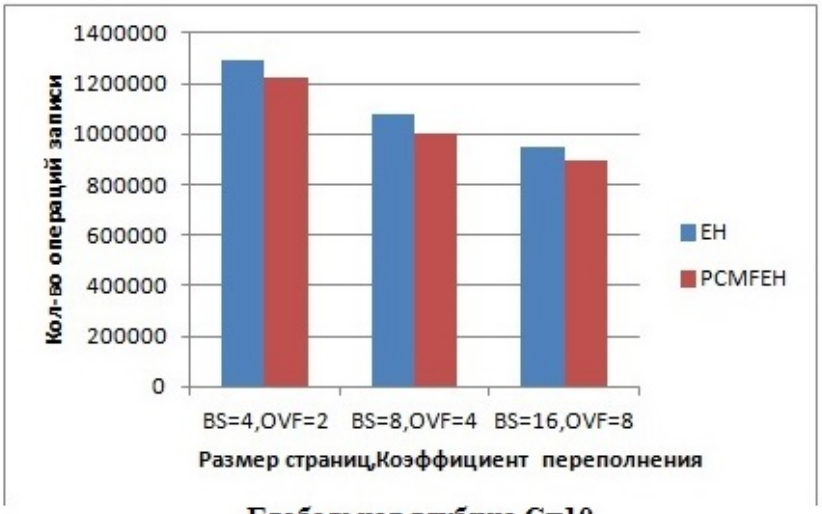


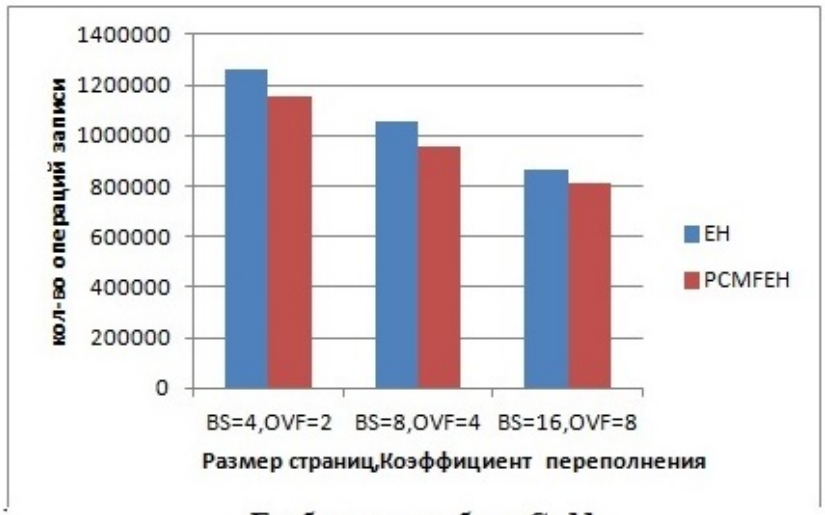
Рис. 12: Зависимость времени поиска от размера страниц (BS) для расширяемого хеширования и модифицированного метода с разными коэффициентами переполнения

Результаты экспериментов показывают, что введенные модификации незначительно влияют на время поиска.

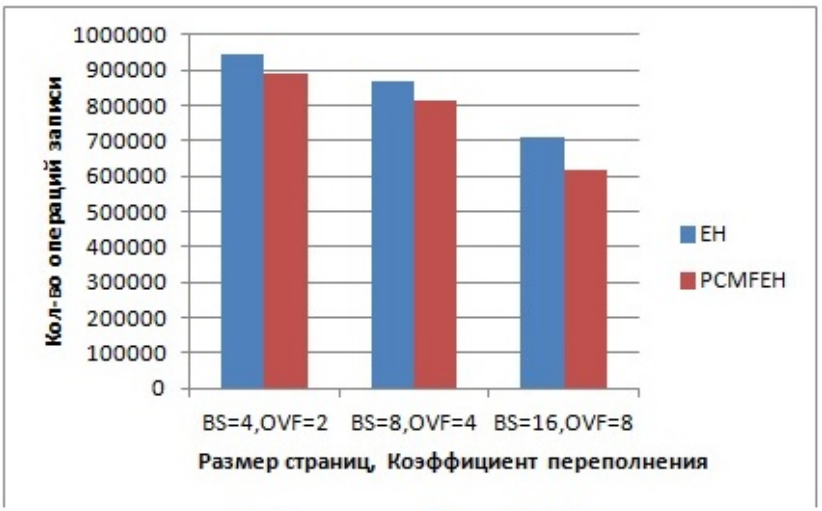
Теперь проверим нашу модификацию на большом объеме данных, а именно будем заполнять структуру 100 000 пар ”ключ-значения”, значение глобальной глубины и размер страниц будем менять. На диаграмме показаны результаты для стандартного расширяемого хеширования и PCMFEH.



Глобальная глубина G=10



Глобальная глубина G=11



Глобальная глубина G=12

Рис. 13: Сравнение количества операций записи расширяемого хеширования (EH) и PCMFEH

Как мы видели ранее, при глобальной глубине больше 8 в модифицированном методе совершается больше операций записи при построении структуры, но это количество не повлияло на результат,что подтвердилось проведенными экспериментами. Модифицированный метод показывает себя лучше стандартного алгоритма расширяемого хеширования и при вставке большого количества пар.(В частности, при большой глобальной глубине)

Получаем, что предложенный метод PCMFEH учитывает ассиметрию между операциями чтения и записи данных в PCM. А именно сокращает количество операций записи при вставке. Но кроме того, как было сказано, PCM имеет ограниченное число перезаписей ячейки. Таким образом для эффективности важно не только уменьшение общего количества операций записи, но и количество перезаписи в ячейке по отдельности.

Для проверки, что и по свойству изнашиваемости PCMFEH лучше, чем стандартное хеширование, будем учитывать операции записи для каждой переменной, затем будем производить вставки пар, как в эксперименте выше, и среди полученых значений для каждой переменной возьмем максимум. Именно по этому значению будем сравнивать.

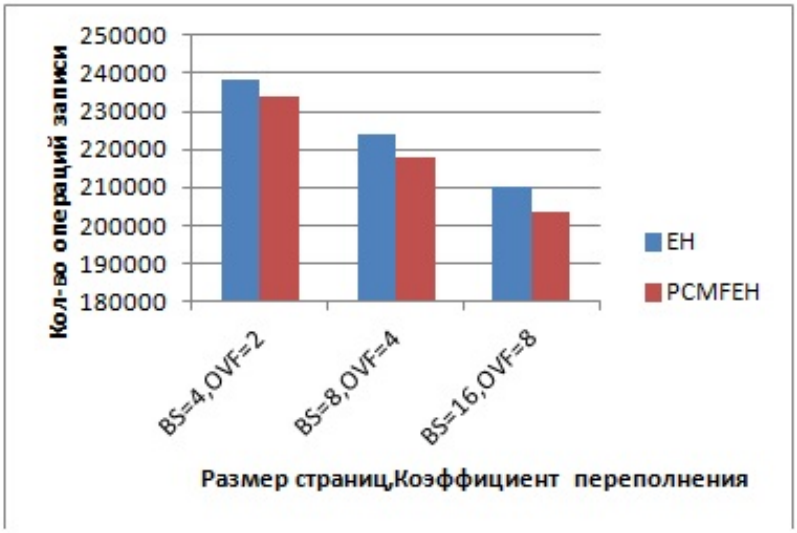


Рис. 14: Сравнение обычного и модифицированного метода по описанному выше параметру

Эксперименты доказали, что модифицированный метод эффективнее, чем стандартное расширяемое хеширование и по этому параметру, влияющему на свойство памяти изнашиваться.

Заключение

В данной работе были исследованы алгоритмы хеширования для памяти типа PCM, который является относительно новым видом памяти, в частности перспективным видом основной памяти. Для этого были изучены его характеристики и свойства, было произведено сравнение PCM с другими типами памяти.

Основным результатом работы является модифицированный алгоритм расширяемого хеширования (PCMFEH). Он представляет собой адаптацию стандартного расширяемого хеширования для памяти с изменением фазового состояния.

Основная идея модификации состоит в разрешении переполнения страницы для уменьшения количества разделений страниц, и как следствие количества перестроений, которые приводят к многочисленным операциям записи. Были произведены эксперименты, показывающие, что измененный алгоритм учитывает проблемы, связанные с операцией записи в памяти PCM, с помощью уменьшения общего количества операций записи в память. Также снижается количество операций записи на ячейки по отдельности, что положительно влияет на сохранность памяти PCM, которая имеет ограниченное количество перезаписи в ячейку.

Список литературы

[1] Architecting Phase Change Memory As a Scalable Dram Alternative / Benjamin C. Lee, Engin Ipek, Onur Mutlu, Doug Burger // SIGARCH Comput. Archit. News. — 2009. — jun. — Vol. 37, no. 3. — P. 2–13.

[2] B^p - $Tree$: A Predictive B^+ - $Tree$ for Reducing Writes on Phase Change Memory / W. Hu, G. Li, J. Ni et al. // IEEE Transactions on Knowledge and Data Engineering. — 2014. — Oct. — Vol. 26, no. 10. — P. 2368–2381.

[3] Chen Shimin, Gibbons Phillip B., Nath Suman. Rethinking Database Algorithms for Phase Change Memory. — 2011. — January.

[4] Chi Ping, Lee Wang-Chien, Xie Yuan. Making B+-tree Efficient in PCM-based Main Memory // Proceedings of the 2014 International Symposium on Low Power Electronics and Design. — ISLPED '14. — 2014. — P. 69–74.

[5] Efficient Tree Indexing for PCM-Based Memory Systems / L. Li, P. Jin, C. Yang, L. Yue // 2015 8th International Conference on Control and Automation (CA). — 2015. — Nov. — P. 46–53.

[6] Extendible Hashing - a Fast Access Method for Dynamic Files / Ronald Fagin, Jurg Nievergelt, Nicholas Pippenger, H. Raymond Strong // ACM Trans. Database Syst. — 1979. — sep. — Vol. 4, no. 3. — P. 315–344.

[7] Herlihy Maurice, Shavit Nir, Tzafrir Moran. Hopscotch Hashing // Proceedings of the 22Nd International Symposium on Distributed Computing. — DISC '08. — 2008. — P. 350–364.

[8] Pagh Rasmus, Rodler Flemming Friche. Cuckoo Hashing // J. Algorithms. — 2004. — may. — Vol. 51, no. 2. — P. 122–144.

[9] Qureshi Moinuddin K., Srinivasan Vijayalakshmi, Rivers Jude A. Scalable High Performance Main Memory System Using Phase-change

Memory Technology // Proceedings of the 36th Annual International Symposium on Computer Architecture. — ISCA '09. — 2009. — P. 24–33.

[10] Revisiting Hash Table Design for Phase Change Memory / Biplob Debnath, Alireza Haghdoost, Asim Kadav et al. // Proceedings of the 3rd Workshop on Interactions of NVM/FLASH with Operating Systems and Workloads. — INFLOW '15. — 2015. — P. 1:1–1:9.